Emre Zeytinoğlu 28190

CS307 PA3 Ride Sharing

This program shows carpooling scenarios for fans of two different teams. It wrote in the C language, multithreading and semaphores are used.

There are several structures and functions in the program:

FanThreadArgs Structure: Contains key information about each fan thread, including the team identifier, team fan count, and team semaphores.

requestRide: Posts a message shows that a fan is looking for a car, requests a ride, and increases the fan count.

joinCar: Modifies fan counts, identifies the car as full, and signals semaphores for the current and other teams when requirements are satisfied for fans to join vehicle.

driveCar: This function signals the current team's semaphores to be ready for motion.

rideFinder: All function that is written above carried out by this function, It includes if and else statements which are stated in the assignment (fans number balance). It handles the ride-finding, car-pooling, and driving processes. handles semaphores for synchronization.

Thread Flow (Pseudocode)

Main Function:

```
function main:
    read teamCounts from command line arguments

    if invalidTeamCounts(teamCounts):
        print "Invalid team counts. The main terminates."
    else:
        initialize teamSemaphores
        initialize globalMutex
        initialize teamFansWaitingCount
```

```
    create threads for Team A and Team B fans
    wait for all threads to finish

    print "The main terminates"

  return 0
```

rideFinder Function:

```
function rideFinder(teamArgs):
  currentTeam = teamArgs.team
  otherTeam = 1 - currentTeam
  isCarFull = 0

  requestRide(teamArgs, currentTeam)

  if teamFansCount[currentTeam] > 1 and teamFansCount[otherTeam] == 2:
    joinCar(teamArgs, currentTeam, otherTeam)
    isCarFull = 1
  else if teamFansCount[currentTeam] == 4:
    driveCar(teamArgs, currentTeam)
    isCarFull = 1
  else:
    waitForCar(teamArgs, currentTeam)

  print "Thread ID: <tid>, Team: <Team>, I have found a spot in a car"

  if isCarFull:
    print "Thread ID: <tid>, Team: <Team>, I am the captain and driving the car"
```

Multithreading

Concurrency: The rideFinder function can be run concurrently by several threads in the program. Both Team A's and Team B's threads are capable of running separately.

Shared Resources: Global variables accessible by several threads, such as globalMutex, teamSemaphores, and teamFansCount, are examples of shared resources. To prevent race situations, proper synchronization is provided.

Synchronization

Mutex (globalMutex): In order to prevent data corruption, it makes sure that only one thread may perform the crucial portion at a time.

Semaphore:
TeamSemaphores is an array of semaphores that functions as a semaphore. Semaphores are used to ensure correct thread coordination by signaling and controlling access to shared resources.

Race Conditions: The likelihood of race conditions is declined by using semaphores (teamSemaphores) and mutexes (globalMutex). To safely access shared resources, threads cooperate.

Summary

The program uses semaphores and Pthreads for synchronization, and it correctly organize the carpooling situation. Through handling shared resource concurrent access, the software offers a reliable and secure simulation.